Arkadaşlar merhaba,

Bir önceki bölümde Modellere giriş yaptık. Nasıl veri yapıları oluşturacağımızı ve veritabanı ile nasıl eş zamanlayacağımızı öğrendik. Oluşturduğumuz veri yapılarının yönetici sayfasında nasıl görüntüleneceğini ve yönetici sayfasının özelleştirilmesi hakkında temek bilgilere değindik. Bu videomuzda temel bilgiler vermeye devam edeceğiz. İleriki konularda ise detaylandıracağız. Bu vidomuzda URL Configuration, Views ve templateler hakkında bilgiler vereceğim.

Kullanıcı tarayıcıyı açtı sitemizin adresini yazdı ve entere bastı. İsteği server Django uygulamasına yönlendiriyor. Django uygulaması istek yapılan adrese ile bizim settings.py den ayarlamış olduğumuz URL Configuration yaptığımız dosyayı karşılaştırır ve hangi URL ile eşleşiyorsa o URL’ye karşılık gelen view metoduna yönlendirir. Biz Django’da URL configurasyonunu stocks/urls.py dosyasında yaparız. Şimdi URL configurasyonu yaptığımız dosyamızı inceleyelim.

Gördüğünüz gibi burada admin arayüzü hazır geliyor. Daha önceki derslerimizde <http://127.0.0.1:8000/admin/> olarak girdiğimizi hatırlarsınız.

Biz buraya example/ isteği yapıldığında başka bir yere yönlendirme yapabiliriz. <http://127.0.0.1:8000/example/> yapıldığında example\_view yönlendirmiş olduk. Tabi yönlendirdikte example\_view nerde ?

urlpatterns = [  
 path('admin/'**,** admin.site.urls)**,** path('example/'**,** views.example\_view)  
]

Gördüğünüz gibi Django’da bulamıyorum ben diye NameError hatası verdi.

**NameError: name 'example\_view' is not defined**

Biz view metodlarımızı views.py dosyasında oluşturuyoruz. Learning uygulamamızda example\_view adında bir tane view metodu oluşturalım. View metodları ilk parametre olarak request değerini alır. Bu parametre ile post, get yapılan verilere ve bir takım diğer verilere ulaşabiliriz.

Html değişkenine ekrana basılacak içeriğimizi girelim. Şimdi bunu ekrana basmamız gerekli bunun için Django http kütüphanesindeki HttpResponse metodunu kullanabiliriz.

from django.http import HttpResponse  
  
  
def example\_view(request):  
 html = "<html><body><h2>MERHABA DÜNYA!</body></head>"  
  
 return HttpResponse(html)

Evet view metodumuz hazır urls.py mizde de tanımlı. Ama hala name error hatası alıyoruz. Çünkü bu metodumuzu urls.py dosyasına import etmedik. İmport edelim ve serverımızı tekrar çalıştıralım.

from learning.views import example\_view

**python manage.py runserver**

Evet serverımız sorunsuz çalıştı. Adresimizi kontrol ettiğimizde ekrana MERHABA DÜNYA! Gelmesi gerekli. Evet oda gelmiş. Yani bir url configürasyonu yaptık ve example/ görürsen example\_view methodunu çalıştır dedik.

Tüm html kodlarını view dosyası içerisine yazmamız çok mantıklı değil. Django’nun MVT yapısına sahip olduğunu söylemiştik. HTML kodları için Template kullanabiliyoruz. View’den templati çağıralım ve ekrana ordan gelen içeriği basalım. Bunun için shortcuts kütüphanesinden render komutunu kullanıyoruz.

from django.shortcuts import render  
  
  
def example\_view(request):  
 return render(request=request**,** template\_name='example/detail.html')

settings.py den template klasörü olarak templates ayarlayalım.

TEMPLATES = [  
 {  
 'BACKEND': 'django.template.backends.django.DjangoTemplates'**,** 'DIRS': ['templates']**,** 'APP\_DIRS': True**,** 'OPTIONS': {  
 'context\_processors': [  
 'django.template.context\_processors.debug'**,** 'django.template.context\_processors.request'**,** 'django.contrib.auth.context\_processors.auth'**,** 'django.contrib.messages.context\_processors.messages'**,** ]**,** }**,** }**,**]

Templates ve example klasörlerini oluşturalım ve içerisine detail.html ekleyelim. İsteğimizi yapalım. Evet template dosyasından yazımız geldi.

Peki biz bütün URL Configürasyonumuzu bu ana dosyamızdamı yapacağız. İstersek yapabiliriz fakat her uygulama için farklı bir url dosyası kullanmak daha derli toplu olacaktır. Yapılan istek uygulamanın adıyla yapılıyorsa direk uygulamaya aktaralım. Yani 127.0.0.1:8000/learning/ adresine yapılan istekler learning/urls.py dosyasındaki URL Configürasyonuna yönlendirilsin. Öncelikle uygulamamız içerisinde urls.py dosyamızı oluşturalım ve bu dosyamızı ana konfigürasyon dosyamıza include edelim. Bunun için urls kütüphanesindeki include metodunu kullanacağız. learning/ gelen istekleri learning.urls yönlendirelim.

Evet yine bir hata ile karşılaştık. Learning.urls dosyaımızın doğru şekilde konfigüre edilmediğini söylüyor. Dosyamızı açalım ve example/ gelen isteği examples metoduna yönlendirelim ve serverımızı tekrar çalıştıralım.

from django.urls import path  
from . import views  
  
urlpatterns = [  
 path('examples/'**,** views.examples)  
]

**python manage.py runserver**

serverımız hatasız olarak çalıştı. learning/examples/ istek yapıldığında metodumuz çalışıyormu kontrol edelim. Evet gördüğünüz gibi ekrana MERHABA DÜNYA! basıldı.

Model dosyalarımızı klasör içerisinde kullanmıştık. View dosyalarımızıda aynı şekilde kullanmak daha derli toplu olmasını sağlayacaktır. View adında klasör oluşturalım ve views.py dosyasını onun içine alalım. İsminide example olarak değiştirelim. klasörün içine \_\_init\_\_.py dosyaımızı oluşturalım ve examples metodunu bırada tanımlayalım.

from .example import examples

learning/examples/ adresini kontrol ettiğimizde çalıştığını görüyoruz. Bu sayede view yapımızı hazırlamış olduk.

İstek geldiğince öncelikle ana urls.py dosyamıza gelecek daha sonra burdan ilgili uygulamanın urls.py dosyasına yönlenecek. Daha sonra uygulamanın ilişkili metodu bulunacak ve çalıştırılacak.

Farklı url patternler oluşturalım ve bunlara karşılık view metodlarını yazarak konuyu detaylandıralım.

Ürünlerimiz olduğunu düşünelim ve bunlar için ürün listesi, ürün detayı, ürün arşivi oluşturacak view ve url yapılarını kuralım. Öncelikle view metodlarımızı oluşturalım. Learning/views içerisine product.py dosyası oluşturalım view metodlarımızı burda yazalım.

# learning/views/product.py

def product(request):  
 html = "<html><body><h2>ÜRÜN DETAY SAYFASI!</body></head>"

return HttpResponse(html)

def products(request):  
 html = "<html><body><h2>ÜRÜN LİSTELEME SAYFASI!</body></head>"

return HttpResponse(html)

def product\_archive(request):  
 html = "<html><body><h2>ÜRÜN ARŞİV SAYFASI!</body></head>"

return HttpResponse(html)

metodlarımızı \_\_init\_\_.py de tanıtalım.

# learning/views/\_\_init\_\_.py

from .product import product, products, product\_archive

products/ yapılan istekleri metodumuza yönlendirelim.

# learning/urls.py

path('examples/'**,** views.examples)**,**

path('product/'**,** views.products)**,**

path('product/detail/'**,** views.product)**,**

path('product/archive/'**,** views.product\_archive)**,**

evet artık product/ istek gönderilince yönlenecek metodlarımızı belirledik. Serverımızı çalıştıralım ve istek yaparak çıktıları kontrol edelim.

product/ yapılan istekleri metodumuza yönelendirdik.Fakat ürünü seçebilmek için bir parametreye ihtiyacımız olacak. Ürün detaylarının gelmesi için hangi ürünün geleceğini bilmemiz gerekir. URL configürasyon dosyamıza ürün bilgileri için birtane parametre ekleyelim

# learning/urls.py

path('examples/'**,** views.examples)**,**

path('product/'**,** views.products)**,**

path('product/detail/<pk>/'**,** views.product)**,**

path('product/archive/'**,** views.product\_archive)**,**

Şimdi bu yazdığımız parametreyi view metodunda yakalamamız gerekli. Product metodumuza argüman olarak pk ekleyelim ve html çıksısına ekleyelim. Eklerken integer olduğu için stinge çevirmemiz gerekli bunun için str() metodunu kullanacağız.

# learning/views/product.py

def product(request**,** pk=None):  
 html = "<html><body><h2>" + str(pk) + " NOLU ÜRÜN DETAY SAYFASI !</body></head>"

return HttpResponse(html)

Evet ekrana çıktı olarak pk’nın geldiğini görüyoruz.. Birde string girerek deneyelim. Stringide bastı. Fakat biz ürün numaralarımızı sadece sayılardan oluşturacağız. Buraya sadece sayı nasıl kabul edebiliriz. Django URL Configürasyonunda Path converters ve regex ile bu alanın sadece sayı almasını sağlayabiliriz.

Öncelikli olarak path converters kullanımına bakalım. Django standart olarak bazı veri tipleri için gelen parametreyi sınırlandırmak için hazır yapı hazırlamıştır.

<int:pk>, <string:pk> diye kullanılır.

* str : sadece string kabul eder ve path seperatörü kabul etmez.
* int : sıfır ve pozitif sayıları kabul eder
* slug : sadece slug kullanımına uygun karakterleri kabul eder
* uuid : uuid formatı kabul eder.
* path : sadece string ve path seperatörü kabul eder.

Şimdi ürünümüzü sadece sayı alması için <int:pk> yaparak bunu halledebiliriz. Değiştirip tekrar deneyelim. Evet string girdiğimizde Page Not Found hatası verdi.

# learning/urls.py

path('examples/'**,** views.examples)**,**

path('product/'**,** views.products)**,**

path('product/detail/<int:pk>/'**,** views.product)**,**

path('product/archive/'**,** views.product\_archive)**,**

İstersek kendi path convertörlerimizi register\_converter() metodu ile tanıtabiliriz. Bu konu hakkında detaylı bilgi için djangonun dökümantasyonundan bilgi alabilirsiniz.

2. yöntem ise regex kullanmaktadır. Bunun detaylarına giremeyeceğim fakat örnek olması açısından sayılar için örnek gösterelim. Regexleri kullanabilmek için path() yerine re\_path() kullanılır. Bunun için re\_path import etmemiz gerekmektedir.

Sadece sayı alması için;

from django.urls import path**,** re\_path  
from . import views  
  
urlpatterns = [  
 re\_path(r'^re\_product/(?P<pk>[0-9])/$'**,** views.product)**,**

]

Son olarak product\_archive metodumuzu düzenleyelim. Tarayıcıdan istek yapılırken parametre olarak yıl ve ay girilsin. Biz bu yıl ve ayda listeye eklenen ürünleri getirelim. Yani biraz önce bir parametre almıştı bu sefer 2 farklı parametre alması gerekli.

Evet çalıştırdığımızda gördüğünüz gibi parametreler olmadığı için hata verdi. View metoduna parametrelerimizi ekleyelim ve HTML çıktısında gelmesini sağlayalım. Değerler int olduğu için stringe çevirmeyi unutmayalım.

def product\_archive(request**,** year=None**,** month=None):  
 html = "<html><body><h2>" + str(year) + " / " + str(month) + " TARİHLİ ÜRÜN ARŞİV SAYFASI!</body></head>"  
 return HttpResponse(html)

Şimdi tüm view metodlarımız için birer tane template doyası oluşturalım ve metodumuzda render() komutu ile onları çağıralım.

# learning/views/product.py  
from django.shortcuts import render  
  
  
def product(request**,** pk=None):  
 return render(request=request**,** template\_name='example/detail.html')  
  
  
def products(request):  
 return render(request=request**,** template\_name='product/list.html')  
  
  
def product\_archive(request**,** year=None**,** month=None):  
 return render(request=request**,** template\_name='product/archive.html')

Serverı çalıştıralım ve kontrol edelim. Evet ürün sayfalarımız ekrana geldi. Fakat parametrelerimizi veya view sayfasında yazdığımız bir değişkeni nasıl template sayfamıza aktaracağız. render() metodunda context argümanını kullaranarak templatelere view içindeki değişkenlerimizi aktarabiliriz.

product sayfasındaki primary keyimizi template sayfasına aktaralım.

def product(request**,** pk=None):  
  
 context = {  
 'pk': pk  
 }  
 return render(request=request**,** template\_name='product/detail.html'**,** context=context)

Dict olarak aktardığımız context değişkenini template’de yakalamak için iki tane süslü parantez arasında eleman ismini yazarız.

Bu sayfa {{ pk }} ürün detay sayfasıdır.

Kontrol ettiğimizde ekrana pk’nın geldiğini görüyoruz. Aynı işlemi archive sayfası içinde yapalım bu sefer 2 tane eleman aktaracağız.

def product\_archive(request**,** year=None**,** month=None):  
 context = {  
 'year': year**,** 'month': month  
 }  
 return render(request=request**,** template\_name='product/archive.html'**,** context=context)

Temada yine dict nesnesinin eleman isimleriyle yakalayabiliriz.

Bu sayfa {{ year }} / {{ month }} için ürün arşiv sayfasıdır.

require\_http\_methods dekoratörünü kullanarak view methodlarına yapılacak olan http metodlarını sınırlandırabiliriz.

product\_archive metodumuzu sadece GET ve POST için sınırlandıralım.

from django.views.decorators.http import require\_http\_methods

@require\_http\_methods(["GET"**,** "POST"])

def product\_archive(request**,** year=None**,** month=None):  
 context = {  
 'year': year**,** 'month': month  
 }  
 return render(request=request**,** template\_name='product/archive.html'**,** context=context)

sadece GET metoduna izin vereceksek require\_GET dekoratörü, sadece POST metoduna izin vereceksen require\_POST dekoratörü kullanabiliriz. Bu metodlarda django.views.decorators.http kütüphanesinden türerler.